

# OPEN SOURCE SOA

Jeff Davis



Unedited Draft

MANNING



Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=416>



**MEAP Edition  
Manning Early Access Program**

Copyright 2008 Manning Publications

For more information on this and other Manning titles go to  
[www.manning.com](http://www.manning.com)

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=416>

## Table of Contents

### **Part 1: History & Principles of SOA**

1. *SOA Essentials*
2. *Open-Source Technology Selections*

### **Part 2: Service Assembly, Enablement & Mediation**

3. *Services Components & Compositions*
4. *Advanced Tuscany*
5. *Enablement using an Enterprise Service Bus*
6. *Web Service Mediation*

### **Part 3: Business Process Composition & Monitoring**

7. *Business Process Management*
8. *Event Stream Processing*

### **Section 4: Metadata Repository & Enterprise Decision Management**

9. *Metadata Repository*
10. *Enterprise Decision Management*

### **Section 5: Case Studies**

11. *Case Study 1: Sales Order Processing*
12. *Case Study 2: TBD (possibly CRM Integration w/Salesforce.com)*

### **Appendixes**

- Appendix A: Setting up Apache Tuscany*
- Appendix B: Setting up ServiceMix*
- Appendix C: Setting up Apache Synapse*
- Appendix D: Setting up jBoss jBPM & BPEL*
- Appendix E: Setting up Esper*
- Appendix F: Setting up Apache Directory Server*
- Appendix G: Setting up JBoss Drools*
- Appendix H: Setting up a Salesforce.com*

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=416>

# *SOA Essentials*

Ponce de León's early quest to find the "Fountain of Youth" in Florida is one of the most frequently told stories of American folklore. While he failed in his journey to find the "healing waters", it turns out that he was in good company. For throughout history there are tails of similar adventures that never materialized. The history of computing bears some resemblance. Every decade or so, there is new "silver bullet" that promises to heal to the problems that have plagued software development in the past. Those problems include protracted development cycles; solutions that fail to achieve expectations; high maintenance costs; and of course, the dreaded cost overruns. The quest is to find a solution that simplifies development and implementation; supports effective reuse of software assets; and leverages the enormous and low-cost computing power now at our fingertips. Some might claim that Software Oriented Architecture, or SOA, is just the latest fad in this illusive quest. However, tangible results have been achieved by those able to successfully implement its principles. Further, it has achieved greater staying power than many earlier alternatives, which does speak something of its merits. Perhaps this is because SOA is a more nebulous concept, and embraces technologies as much as it does principles and guidelines – thus refuting its benefits becomes more difficult.

Until recently, achieving a technology infrastructure capable of sustaining an SOA generally required purchasing expensive commercial products. This was especially true if an enterprise desired a well-integrated and comprehensive solution. While several early SOA-related open source products were introduced, they tended to focus on specific, niche areas. For example, Apache Axis was first introduced in 2004, and became a widely adopted web services toolkit for Java. As we will discover, however, web services represent only a piece of the SOA puzzle. Fast-forward to 2008 and we now see commercially competitive open source products across the entire SOA product spectrum. The challenge now for an SOA architect wanting to use open source is how to select among the bewildering number of competing products. Even more challenging is how to integrate them.

The goal of this book is to help the reader identify the core technologies that constitute an SOA and which open source technologies can be used to build a complete SOA platform. The emphasis will be on how to integrate these core technologies into a compelling solution that is comparable in breadth and depth to the expensive offerings provided by the commercial vendors. SOA is now attainable for even the smallest of enterprises using high-quality open source software. The book will present a technology blueprint for open source SOA. Of course, thanks to the plethora of quality open source solutions, you can naturally swap out the solutions I'm advocating with those you deem appropriate.

Before jumping head-strong into the technology stack, let's first establish some context for where SOA originated, and develop a common understanding of what it is.

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

## 1.1 Brief History of Distributed Computing

The mainframe systems of the 60s and 70s, such as the IBM System/360 series, rarely communicated with each other. Indeed, one of the main selling points of a mainframe was that it would provide you everything necessary to perform the computing functions of a business. When communications were required, it usually amounted to. Over time, though, real-time access between systems became necessary, especially as the number of systems within an organization multiplied. This was especially true in financial markets, where trading required real-time transactional settlements that often spanned across companies. Initially, this was accomplished through low-level socket communications. Usually written in assembly language or C, socket programming was complex, and required a deep understanding of the underlying network protocols. Over-time, protocols such as NFS and FTP emerged that abstracted out the complexity of sockets. Over time, companies such as Tibco emerged that developed “middleware” software explicitly designed to facilitate messaging and communications between servers. Eventually, the ability to create distributed applications became feasible through the development of Remote Procedure Calls, or RPCs. This enabled discrete functions to be performed by remote computers as though they were running locally. As Sun Microsystems’ slogan puts it, the “The Network is the Computer”.

By the 1980s, personal computers had exploded onto the scene, and developers were seeking more effective ways to leverage the computing power of the desktop. As the price of hardware came down, the number of servers within the enterprise also increased exponentially. These trends, coupled with the growing maturity of RPC, lead to two important advances in distributed computing:

**Common Object Request Broker Architecture (CORBA).** Originated in 1991 as a means for standardizing distributed execution of programming functions, the first several releases only supported the C programming language. Adoption was slow, as commercial implementations were expensive and the ambiguities within the specification made for significant incompatibilities between vendor products. The 2.0 release in 1998 was significant in that it supported several additional language mappings, and addressed many of the shortfalls present in the earlier standards. However, the advent of Java, which dramatically simplified distributed computing through RMI, and finally, through XML, has largely led to the demise of CORBA (at least in new implementations).

**Distributed Computing Object Model (DCOM).** DCOM is a proprietary Microsoft technology that was largely motivated as a response to CORBA. The first implementations appeared 1993. While successful within the Microsoft world, the proprietary nature obviously limited its appeal. The wider enterprise class of applications that were emerging at the time, namely Enterprise Resource Planning (ERP) systems, generally use non-Microsoft technologies.

By the late 1990s and the widespread adoption of the internet, companies began recognizing the benefits of extending their computing platform to partners and customers. Prior to this, communications between organizations were very expensive, having to rely on leased lines (private circuits). This made it impractical except for the largest of enterprises. Unfortunately, using CORBA or DCOM over the internet proved to be very challenging. In part, this was due to networking restrictions imposed by firewalls that only permitted HTTP traffic (necessary for browser and webserver communications). Another reason was that neither CORBA nor DCOM commanded dominant market share, so companies attempting communication links often had competing technologies.

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=416>

When the Simple Object Access Protocol, or SOAP first arrived (2000/1) and was touted as a panacea due to its interoperable reliance on XML. It was largely envisioned as an RPC alternative to CORBA and DCOM. Since RPCs were the predominate model for distributed computing, it naturally followed that SOAP was originally used in a similar capacity. However, RPC-based solutions, regardless of their technology platform, proved nettlesome.

### **1.1.1 Problems Related to RPC-Based Solutions**

While RPC-based distributed computing was no doubt a substantial improvement over earlier lower-level socket based communications, it suffered from several limitations:

- Tight coupling between local and remote systems requires significant bandwidth demands. Repeated RPC calls from a client to server can generate substantial network load.
- The fine-grained nature of RPC requires a highly predicable network. Unpredictable latency, a hallmark of internet-based communications, is unacceptable for RPC-based solutions.
- RPCs data type support, which aims to provide complete support for all native data types (arrays, strings, integers etc), becomes very challenging when attempting to bridge between incompatible languages, such as C++ and Java. Often, incompatibilities result, greatly complicating its use.

SOAP RPC style messages also suffered from the same inherent limitations as those mentioned above. Fortunately, SOAP offers alternative message styles that overcome these shortcomings.

### **1.1.2 Understanding SOAP's Messaging Styles**

In addition to the RPC style SOAP messaging, the founders of the standard also had the foresight to create what is known as the document style SOAP message. As pointed out earlier, the RPC style is for creating tightly coupled, distributed applications where a running program on one machine can rather transparently invoked a function on a remote machine. The intention with RPC is to treat the remote functional in the same manner as you would a local one, without having to dwell on the mechanics of the network connectivity. For example, a conventional client/server application could utilize SOAP RPC style messaging for its communication protocol.

Document style, on the other hand, was envisioned more as a means for application-to-application messaging, perhaps between business partners. In other words, it was intended for more “loosely-coupled” integrations such as document or data transfers.

The differences between the two styles are defined within the SOAP standard, and are reflected in the Web Service Definition Language (WSDL) interface specification that describe a given service. RPC style SOAP messages do not fully utilize XML schema to define their message content. Instead, simple XSD schema types are used to express the method parameters. This WSDL fragment illustrates how it is defined:

```
<message name="myRequestMessage">  
  <part name="param1" type="xsd:int"/>
```

Please post comments or corrections to the Author online forum at

<http://www.manning-sandbox.com/forum.jspa?forumID=416>

```

    <part name="param2" type="xsd:float" />
</message>
<message name="empty" />

<portType name="port">
  <operation name="myMethod">
    <input message="myRequestMessage" />
    <output message="myEmptyResponse" />
  </operation>
</portType>

```

Notice that the operation defined as “myMethod” has, as its input, a message definition called “myRequestMessage” that identifies two parameters, “param1” and “param2”. The data types for these parameters are defined through XSD types. Notice there is no XML Schema used, and thus support for more sophisticated validation checking is also not possible. A SOAP call using an RPC/literal style for the above method would resemble:

```

<soap:envelope>
  <soap:body>
    <myMethod>
      <param1>5</param1>
      <param2>5.0</param2>
    </myMethod>
  </soap:body>
</soap:envelope>

```

The operation method is identified as the first element within the `soap:body` element, which in this case is `myMethod`. The only distinction between the RPC/literal and RPC/encoded styles is that the later includes the data type within the parameter (in general, the use of RPC/encoded style is discouraged).

```

<soap:envelope>
  <soap:body>
    <myMethod>
      <param1 type="xsd:int">5</param1>
      <param2 type="xsd:float">5.0</param2>
    </myMethod>
  </soap:body>
</soap:envelope>

```

In light of the aforementioned issues with RPC-style computing, many of the most vocal advocates of SOAP now encourage the use of Document/literal-style SOAP. This style was primarily designed for the exchange of XML documents. For example, an operation used to receive a sales or purchase order xml document. Coincidentally, as I will talk about later, this also is consistent with an SOA goals of exposing primarily course-grained services which tend to better facilitate reuse. Since this style was explicitly designed for the exchange of complex XML documents, it uses XML Schema to define the structure and validation requirements of the document. How would a Document/literal operation be defined to resemble the previous RPC/literal example? The first thing to notice is

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

that XML-Schema is now used to define the input and output messages, as shown in the WSDL fragment shown in figure 1.1:

```
<types>
  <schema>
    <element name="myMethod">
      <complexType>
        <sequence>
          <element name="param1" type="xsd:int"/>
          <element name="param2" type="xsd:float"/>
        </sequence>
      </complexType>
    </element>
    <element name="myMethodResponse">
      <complexType/>
    </element>
  </schema>
</types>

<message name="myRequestMessage">
  <part name="parameters" element="myMethod"/>
</message>

<message name="myEmptyResponse">
  <part name="parameters" element="myMethodResponse"/>
</message>

<portType name="port">
  <operation name="myMethod">
    <input message="myRequestMessage"/>
    <output message="myEmptyResponse"/>
  </operation>
</portType>
```

**Figure 1.1 Relationship between operation, message part and schema definition**

The request (inbound) message specifies a message name of `myRequestMessage`, which, in turn, uses the embedded schema element called `myMethod`. As a `complexType`, it then defines the two parameter values.

The inbound SOAP message would then resemble:

```
<soap:envelope>
  <soap:body>
    <myMethod>
      <param1>5</param1>
      <param2>5.0</param2>
    </myMethod>
  </soap:body>
</soap:envelope>
```

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

As you can see, it rather resembles the RPC/literal example. In part, this is because we “wrapped” the inbound message within the myInput element. This element could have been omitted through the definition, but this technique, popularized by Microsoft’s .NET platform, is common, as it improves human readability.

While there may seem to be little advantage in using Document/literal in this case (especially given terse nature of the XML document being exchanged), the advantages do become clear if we wish to start introducing more data validation. For example, let’s say that param2 should only be a value between 0 and 100. This can be accomplished by modifying the schema in the following manner:

```
<element name="myMethod">
  <complexType>
    <sequence>
      <element name="param1" type="int"/>
      <element name="param2" type="param2Type"/>
    </sequence>
  </complexType>
</element>

<simpleType name="param2Type">
  <restriction base="float">
    <minExclusive value="0"/>
    <maxExclusive value="100"/>
  </restriction>
</simpleType>
```

The “restriction” element of the param2Type simpleType provides such capabilities. You can craft fairly elaborate validation rules within XML Schema.

Another advantage of the Document vs RPC method is that we are not attempting to map the SOAP message to a particular language implementation. So, after the initial flirtation with RPC-based web services, a coalescing of support emerged for the document style SOAP messaging. Microsoft was an early proponent of the document style, and Sun likewise embraced it in more recent version of its Web Services Developers Pack (WSDP). Web services became viewed as a panacea to achieving SOA. After all, a lynchpin of SOA is the service, and a service requires 3 fundamental aspects: implementation; elementary access details; and a contract [SOA: Concepts, BPEL and SCA for the Business Developer. Ben Margolis w/Joseph Sharpe, pg 197]. A SOAP-based web service, with its reliance on the WSDL standard, appeared to address all three. The implementation is the coding of the service functionality; the access details and contract are addressed within the WSDL as the port type and XML schema used for document-style messaging. So, if you simply exposed all your internal components as SOAP-based services, you then have the foundation by which you can a) readily reuse the services; and b) combine the services into higher-level business processes – characteristics that eventually would become cornerstones of SOA. So, what exactly is SOA?

### **1.1.3 Advent of Service Oriented Architecture**

The concepts which today are associated with SOA began to emerge with the widespread adoption of the internet, and more specifically, HTTP. By 2003, Roy Shulte of the Gartner Group had coined the term SOA, and it quickly

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

became ubiquitous. What it was, exactly, remained somewhat difficult to quantify. Through time, some commonalities appeared in the various definitions:

“Contemporary SOA represents an open, agile extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services”. [Erl2005]

1.

“Service-Oriented Architecture is an IT strategy that organizes the discrete functions contained in enterprise applications into interoperable, standards-based services that can be combined and reused quickly to meet business needs.” [BEA]

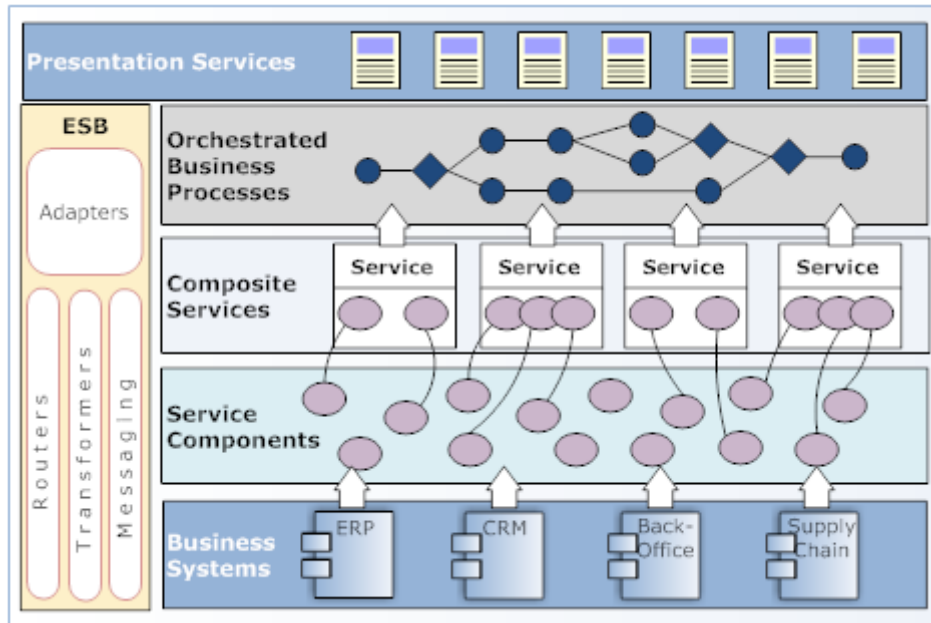
As you can see, the common theme is obviously the notion of discrete, reusable business services that can be used to construct new and novel business processes or applications. As pointed out previously, however, many past component-based frameworks attempted similar objectives. What distinguishes these approaches from the newer SOA? There are several:

- As discussed earlier, CORBA, EJB and DCOM are all based on remote-procedure call (RPC) technologies. In many ways, this is the exact opposite of SOA, since it introduces highly coupled solutions by way of using distributed objects and remote functions. SOA, on the other hand, specifically encourages loosely coupled services.
- In the case of EJB and DCOM, they are both tied to specific platforms and were thus not interoperable. Unless a homogenous environment existed (rare in today’s enterprises, which are often grown through acquisition), the benefits from them could not be easily achieved. SOA-based web services were designed with interoperability in mind.
- Complexity. CORBA, EJB, and to a lesser degree DCOM, were complicated technologies that often required commercial products to implement. SOA can be introduced using a multitude of off-the-shelf, open source technologies.
- SOA relies upon XML as the underlying data representation, unlike the others which used proprietary, binary-based objects. XML’s popularity is undeniable, in part because it is easy to understand and generate.

Another distinction between an SOA and earlier component-based technologies is that an SOA is more than technology per se, but also constitutes a set of principles or guidelines. This includes notions such as governance, service-level agreements; meta-data definitions, and registries. These topics will be addressed in greater detail in the sections that follow.

So what does an SOA resemble conceptually? Figure 1.2 depicts the interplay between the backend systems, exposed services, and orchestrated business processes.

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>



**Figure 1.2** Illustration of an SOA Environment

As you can see, service components represent the layer atop the enterprise business systems/applications. These components allow the layers above to interact with these systems. The composite services layer represents more course-grained services that are comprised from two or more individual components. For example, a “createPO” composite service may include integrating finer-grained services such as “createCustomer”, “createPOHeader”, and “createPOLineItems”. The composite services, in turn, can then be called by higher-level orchestrations, such as one for processing orders placed through a web site.

What is interesting is that, in many respects, SOA is a significant departure from older distributed computing models, which centered the exchange of distributed objects and remote functions. SOA instead emphasizes a loosely coupled affiliation of services that are largely autonomous in nature.

## **1.2 The Promise of Web Services for Delivering SOA**

The SOAP standard, with its reliance on WSDLs, appeared to address many of the fundamental requirements of a SOA. That being the case, SOA, in many individual’s eyes, became rather synonymous with web services. The major platform vendors such as Sun, IBM, Microsoft, BEA and JBoss developed tools that greatly facilitated the creation of SOAP-based web services. Companies began to eagerly undertake proof-of-concept initiatives to scope out the level of effort required to participate in this new paradigm. Web commerce vendors were some of the earliest proponents of exposing their API through SOAP, with eBay and Amazon.com leading the way (more than 240,000 people have participated in Amazon Web Services). “Software As A Service” (SAAS) vendors such as Salesforce emerged that greatly leveraged on the promise of web services. Indeed, Salesforce became the poster boy for what the next generation of software was touted to become.

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

Within organizations, however, the challenge of exposing core business functionality as web services turned out to be daunting. Simply exposing existing objects and methods as web services often proved ill-advised -- to do so simply embraces the RPC-model of distributed computing, not the SOA principles of loosely coupled, autonomous services. Instead, façade patterns or wrappers were often devised to create the desired web services. This often entailed writing significant amounts of new code, which contrasted with the heady promises made by vendors. The challenges were compounded by the vast number of choices that were available, even within a particular language environment. In the Java world alone, there were a bewildering number of choices for creating web services: Apache Axis (and Axis 2); Java Web Services Developers Pack; Spring-WS, JBossWS, and XFire. And these are just the open source products! Knowing which technology to use alone required significant investment.

Other factors also served to dampen the interest in SOAP web services. The perceived complexity of the various WS-\* standards lead to a movement to simply use XML-over-HTTP, as is the basis for REST-based web services (see Wikipedia for an excellent write-up of how REST contrasts with SOAP). The nomenclature found in the WSDL specification, such as port types and bindings is alien to many developers and strikes them as overly convoluted, especially for simple services.

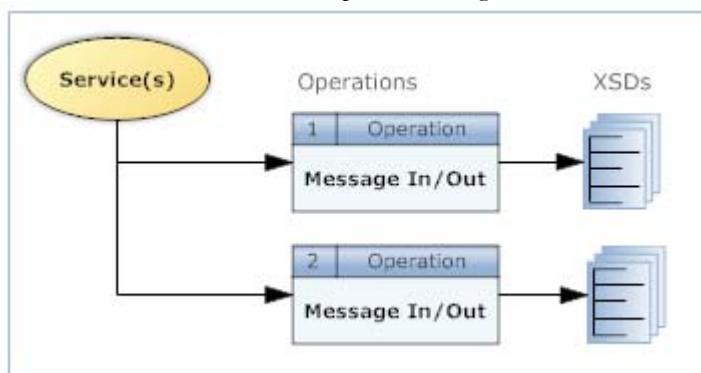
The early enthusiasm for SOAP-based web service as the springboard for SOA began to wane.

### **1.3 Core Characteristics of SOA**

As it turns out, achieving SOA requires more than SOAP-based web services. The characteristics of SOA transcend a particular technology. SOA is an amalgamation of technologies, patterns and practices, the most important of which are addressed next.

#### **1.3.1 Service Interface/Contract**

Services must have a well-defined interface or contract. A contract is the complete specification of a service between a service provider and a specific consumer. It should also exist in a form that is readily digestible by possible clients. This contract should identify what operations are available through the service; define the data requirements for any exchanged information; and detail how the service can be invoked. A good example of how such a contract can be crafted can be found in a WSDL. Apart from describing which operations are available through a given network “endpoint”, it also incorporates XML Schema support to describe the XML message format for each of the service operations. Figure 1.3 illustrates the relationship between WSDL and XML Schema:



Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

### **Figure 1.3 WSDL Usage of XML Schema for defining the specification of an operation**

Multiple operations can be defined, each of which can have its own schema definition associated with it. While the WSDL nomenclature can be very confusing (particularly the 1.1 specification, with its rather arcane concepts of ports and bindings), it has, arguably, been the most successful means for defining what constitutes an interface and contract for a service. Commercial vendors, in particular, have created advanced tooling within their platforms that can parse and introspect WSDLs for code generation and service mapping. The WSDL 2.0 specification is intended to simplify the learning curve and further advance its adoption.

One of the early criticisms of the WSDL specification was that the specific service endpoint was “hardwired” into the specification. This limitation was largely addressed in the WS-Addressing standard, which has achieved widespread adoption. It supports dynamic endpoint addressing by including the addressing information within the body of the SOAP XML message, and not “outside” of it within the HTTP SoapAction HTTP header. The endpoint reference contained within the WS-Addressing block could also be a logic network location, not a physical one. This enables more complex load-balancing and clustering typologies to be supported. We address the issue of why such “service transparency” is beneficial next.

### **1.3.2 Service Transparency**

Service transparency pertains to the ability to call a service without specific awareness of its physical endpoint within the network. The perils of using direct physical endpoints can be found in recent history. Nearly all enterprise systems began offering significant API support for their products by the mid-1990s. This allowed clients to begin to tap into the functionality and business rules of the systems relatively easily. One of the most immediate, and undesirable, consequences of doing this was the introduction of point-to-point interfaces. Before long, you began seeing connectivity maps that resemble figure 1.4.

**Error! Objects cannot be created from editing field codes.**

**Figure 1.4 Example of how point-to-point connections greatly complicate service integration.**

An environment punctuated by such point-to-point connections quickly becomes untenable to maintain and extremely brittle. A change in something as simple as endpoint connection string or URI can break a multitude of applications, perhaps even unknowingly. For example, in figure 1.4, imagine if the CRM system’s network address changed – a multitude of other apps would immediately break.

An ESB is often touted as the savior for avoiding the proliferation of such point-to-point connections, since its messaging bus can act as a conduit for channeling messages to the appropriate endpoint location. It no doubt performs such functionality admirably, but this also can be accomplished through a simple service mediator or proxy. The scenario depicted in figure 1.4 could then be transformed to the one shown in figure 1.5.

**Error! Objects cannot be created from editing field codes.**

**Figure 1.5 Example of mediator or proxy-based service endpoint environment**

Obviously, figure 1.5 is an improvement over figure 1.4. No longer does the client application or API user have to explicitly identify the specific endpoint location for a given service call. Instead, all service calls are directed to the proxy or gateway, which, in turn, forwards the message to the appropriate endpoint destination. If an endpoint address thus changes, only the proxy configuration will be required to be changed.

Please post comments or corrections to the Author online forum at

<http://www.manning-sandbox.com/forum.jspa?forumID=416>

The WS-Addressing specification, one of the earlier and most well supported of the WS-\* standards, defines an in-message means for defining the desired endpoint or action for SOAP-based web services. It is significant in that, without it, only the transport protocol (typically HTTP) contains the routing rules. WS-Addressing supports the use of logic message destinations, which would leave the actual physical destination to be determined by a service mediator.

Until fairly recently, there weren't any true open source web service proxy solutions available. However, Apache Synapse, although sometimes positioned as an ESB, is designed largely with this capability in mind. It supports outstanding proxy capabilities, and can also serve as a protocol switcher. For instance, it can be easily configured to receive a SOAP HTTP message and deposit it for internal consumption by a Java JMS queue. Synapse will be covered in depth in later sections of this book.

### **1.3.3 Service Loose-Coupling & Statelessness**

Simply exposing a service as a SOAP-based web service, defined by a WSDL, does not, by itself, constitute service enablement. A key consideration is also whether the service is sufficiently self-contained so that it could be considered stand-alone. This is sometimes referred to as the level of “service coupling”. For example, let's assume that we want to create a new service to add a new Customer into your company's CRM system. If, in order to use the service you must include CRM-specific identifiers such as OrganizationId, you have now predicated the use of that service on having a prior understanding of the internals of the CRM. This can greatly complicate the use of the service by potential consumers, and may limit its audience potential. In this case, it would be preferable to create a composite service that performs the OrganizationId lookup first, and then followed by the call to insert the new Customer.

Related to the above is the issue of granularity. This pertains to the scope of functionality that is addressed by the service. For instance, a “fine-grained” service may resemble something like “addCustomerAddress”, whereas a “course-grained” service is more akin too “addCustomer”. The preponderance of literature advocates the use of “course-grained” services, in part for performance reasons as well as convenience. If the desire to is to add new Customer to your CRM system is the objective, calling a single service with a large XML payload is obviously preferable than having to chain together a multitude of lower-level service calls. That said, maximizing reusability may sometimes warrant the construction of finer-grained services. In our example above, having the ability to “addCustomerAddress” can be used in a variety of cases not limited to just creating a new Customer. Indeed, composite services that are courser-grained in function can then be crafted based-upon the lower-level services.

Lastly, if possible, a service should be stateless. What would be an example of a stateful service? One might imagine a service that includes a “validation” operation that first must be called prior to the actual action operation. If successful, the validation call would return a unique identifier. The action operation would then require that validation Id as its input. In this scenario, the data input from the validation call would be stored in a session state awaiting a subsequent call to perform the desired activity. While this solution avoids forcing the client user to resubmitted the complete data-set twice (one for the operation, the other for the action), it introduces a lot of additional complexity for the service designer. In particular, scalability can be adversely impacted, as the application server must preserve session state and manage the expiration of unused sessions. Performance

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=416>

management is complicated if appliance-based load-balancing is being used, as it must pin the session calls to specific application servers (software-clustering can overcome this, but it introduces its own challenges).

In the above scenario, statefulness can be avoided by requiring the client to again send all relevant data when making the action call, along with the validation id retrieved from the validation call. The validation id would be persisted in a database and provided a timestamp. The action call would have to take place within a given number of minutes before the validation id became invalidated.

### **1.3.4 Service Composition**

One of the main objectives of an SOA is the ability to generate composite services and/or orchestrations using service components as the building blocks. A “compositionable service” is largely a function of how well designed it is to participate in such a role. As was illustrated in figure 1.2, there are two general types of composite services. The first type, which could be classified as “simple” or “primitive”, simply wraps one or more lower-level services together into a more coarse-grained operation. This can usually be accomplished by defining a simple data-flow which stitches together services and then exposes the new functionality itself as a new service. Another reason may be to simply impose a new service contract for an existing service without desiring to make any underlying changes to the target endpoint. In any case, the underlying service or services participating in the simple composition must adhere to these attributes we’ve already addressed (and some of which will follow). They include a well-defined service contract; stateless implementation; minimal or loose coupling; and consistent and reliable performance and availability (see the next section). A composite service should be no different, from an interface standpoint, than a lower-level service. This is depicted in figure 1.6.

**Error! Objects cannot be created from editing field codes.**

**Figure 1.6 Illustration of composite service being added to existing catalog of services**

The second type of composite services are the “complex” or workflow-type business processes. This is often referred to as Business Process Management (BPM). These processes are generally multi-step creations that may optionally include long-running transactions. The WS-BPEL set of standards defines an XML-based language for describing a sequence flow of activities. This is accomplished by way of a rich set of nodes that can be used for routing, event handling, exception management (compensation) and flow control. The core WS-BPEL standard is tailored for working with SOAP-based web services. Because of this orientation, the entry point for invoking a WS-BPEL process is most typically a SOAP web service (other possibilities may include a timer service, for example). This can be either a blessing or a curse, depending upon whether SOAP services are a standard within your environment.

How does a composite service author have visibility into which services are available for use when constructing such processes? This is the role of the service registry, covered next.

### **1.3.5 Service Registry & Publication**

Unlike in the movie Field of Dreams, “if you build it, they will come” doesn’t apply to services. Clients must be aware of the existence of a service if they are expected to use it. Not only that, services must include a specification or contract which clearly identifies input, outputs, faults and available operations. The web services WSDL specification is the closest and most well-adopted solution for service reflection. The Universal Description, Please post comments or corrections to the Author online forum at

<http://www.manning-sandbox.com/forum.jspa?forumID=416>

Discovery and Integration (UDDI) standard was intended as a platform-independent registry for web services. It can be used as both a private or public registry. Further, using the UDDI API, a client could theoretically, at least, “discover” services and bind to them. Unfortunately, UDDI suffered from an arcane and complex nomenclature, and its dynamic discovery features were myopic and naïve. Today, there are relatively few enterprise customers using UDDI, and fewer still public registries. For all practical purposes, this is a dead standard, with unfortunately no replacement or alternative in sight.

The failure of UDDI doesn’t obviate the need for a registry, and most companies have instead devised a variety of alternatives. For SOAP-based web services, a comprehensive WSDL can often be adequate. It can list all of the available services and operations. Others have used simple database or LDAP applications to capture service registry information. Simply storing a catalog of services and their descriptions and endpoints in a Wiki may suffice for many companies.

Now that we’ve identified some of the core characteristics of SOA, we will now turn our attention to how those higher-level objectives can be decomposed into specific technologies that, when combined, can comprise a complete SOA technology platform.

## 1.4 Building an SOA Technology Platform

As pointed out previously, it is a mistake to assume that SOA is all about technology choices. Issues like governance, quality of service etc., are all major contributors in crafting a complete SOA. That said, my intention is to focus on the technical aspects, as the other areas largely fall outside the scope of this book. Figure 1.7 depicts the various technologies that constitute an SOA technology platform. In turn, each will be discussed in greater detail.

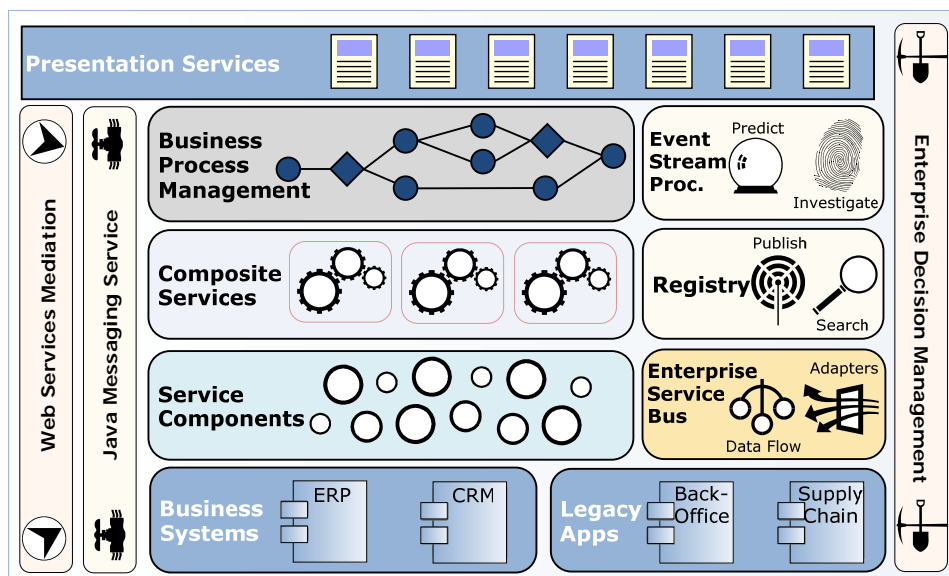


Figure 1.7 SOA Technology Platform

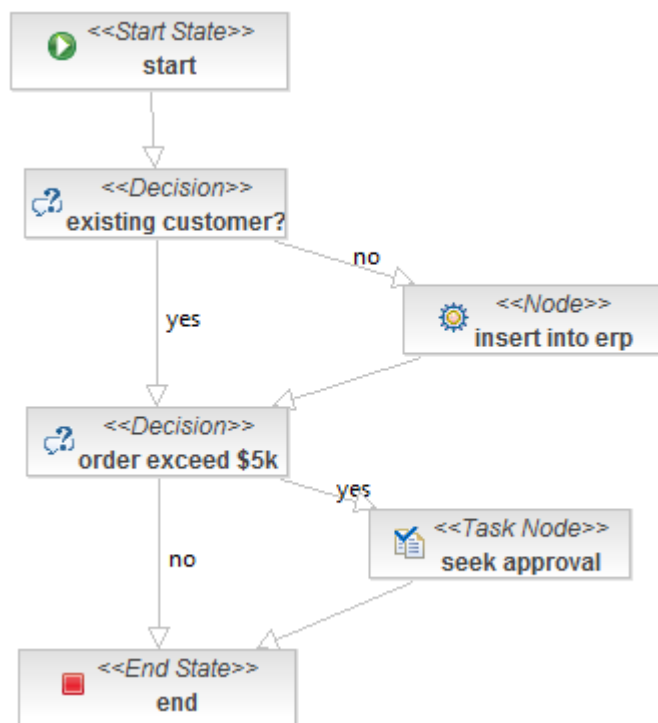
Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

## 1.4.1 Business Process Management

Business Process Management, or BPM, is considered a set of technologies that enables a company to build, usually through visual workflow steps, executable processes that span across multiple organizations or systems. Or, more optimistically stated by Howard Smith and Peter Finger, “BPM doesn’t speed up applications development; it eliminates the need for it” [SmithFinger]. This is because business applications, in this historical context, create stovepipes that are separated by function, time and the data they use. The “process” in BPM refers to a more holistic view of the enterprise, which incorporates employees, partners, customers, systems, applications and databases. This also serves to extract the full value of these existing assets in ways heretofore not possible.

For a system to participate in an BPM process, services or functionality must be made externally accessible. This is why SOA is often considered a pre-requisite for BPM, since SOA is fundamentally about exposing services in a way that enables them to participate in higher-level collaborations.

Figure 1.8 depicts a simple BPM orchestration using the JBoss jBPM suite:



**Figure 1.8** Simple jBPM business process example.

Unlike WS-BPEL, which is intended to be platform neutral (i.e., neither .NET nor Java based etc), jBPM is unabashedly Java, which make using it in Java-oriented environments very straightforward. Chapter 6 will cover jBPM in greater detail.

Like a lot of IT doctrines, while BPM maybe a fairly new buzzword, the concepts have been around for a considerable time. Many of them originated with workflow applications, which enabled the creation of multistep

Please post comments or corrections to the Author online forum at

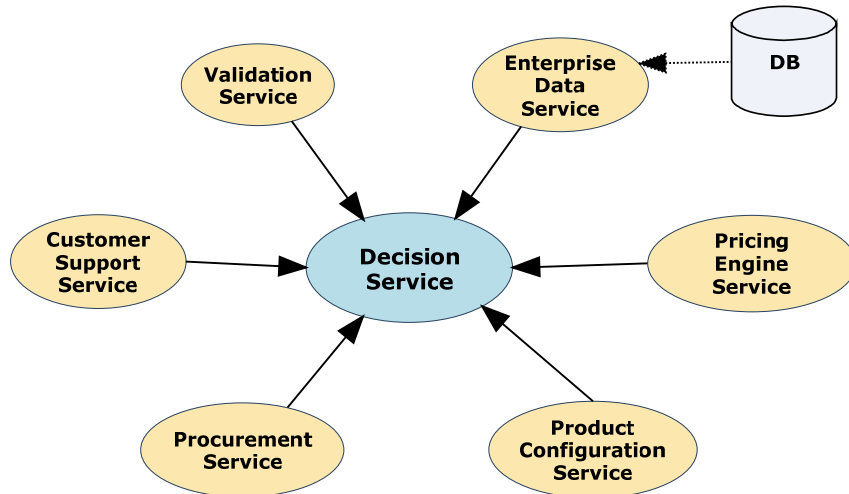
<http://www.manning-sandbox.com/forum.jspa?forumID=416>

models that could be translated into executable software. One important distinction was that prior workflow solutions tended to focus most exclusively on “peopleware”, with steps representing activities to be performed by individuals. Less emphasis was placed on processes that spanned between different systems, largely because there was often no effective way of communicated to them via an API. The advent of XML-based web services has made such capabilities feasible. Theoretically, at least, BPM allows business users to design applications using a lego-like approach – piecing together software services one-upon-another to build a new higher-level solution. In reality, it is obvious not quite so simple, but skilled business analysts can use the visual design and simulation tools for rapid prototyping. These design primitives can also be highly effective at conveying system requirements.

The fundamental impetus behind BPM is cost-savings and improved business agility. As Vivek Ranadive notes, “The goal of BPM is to improve an organization's business processes by making them more efficient, more effective and more capable of adapting to an ever-changing environment.” [Ranadive] Integrating many disparate systems and linking individuals across organizational boundaries into coherent processes can naturally result in significant ROI. A useful byproduct of such efforts is improved reporting and management visibility. Agility, or the ability of a company to quickly react to changes in the marketplace, is improved by enabling new business processes to be quickly created, using existing investments in technology.

### **1.4.2 Enterprise Decision Management**

An Enterprise Decision Management (EDM) system incorporates a Business Rule Engine (BRE) for executing defined business rules and a Business Rules Management System (BRMS) for managing the rules. What exactly is a business rule? It is a statement, written in a manner easily digestible by those within the business, which makes an assertion about some aspect of how the business should function. For example, a company’s policy for when to extend credit is based on certain business rules, such as whether the client has a Dun & Bradstreet number and has been in business for x-amount of years. Such rules permeate most historical applications, where literally thousands of them maybe defined with the application code. Unfortunately, when they are within application code, modifying the rules to reflect changing business requirements is costly and time-consuming. A rules-based system, or BRMS, attempts to cleanly separate such rules from program code. The rules can then be expressed in a language the business user can understand and easily modify without having to resort to application development changes. This also serves to make business rules an “enterprise asset” that represents the very life-blood of an organization. Figure 1.9 illustrates how a centralized decision service can be used by services and applications.



**Figure 1.9 Illustration of how centralized decision service can be used by other services/applications**

One of the biggest challenges when building applications is bridging the knowledge gap that exists between the Subject Matter Experts (SME) who have an intimate understanding of the business, and the developers who often possess only a cursory awareness (and sometimes desire no more than that). Developers are faced with translating business requirements into abstract representations in code. This gap is often responsible for the disappointing results that too often surround the roll-out of new applications. As Taylor and Raden note: "Embedding business expertise in the system is hard because those who understand the business can't code, and those who understand the code don't run the business" [TaylorRaden] It is a commonly held fact that typically 75% of a system's costs are attributable to maintenance, which often includes the modification of built-in business rules. While BRMS are nothing new (dating back to the 1980s with the introduction of "expert systems"), it wasn't until the advent of SOA where integration between the BRMS and the applications using it became cost-effective. With SOA, business rules can be easily exposed via web services.

What differentiates a BRMS from an EDM? To be honest, it's probably mostly semantics, but EDM does emphasize centralized management of ALL business rules, include those considered operational, which may range in the thousands for a given company. According to Taylor and Raden, this includes heretofore "hidden" decisions that permeate a company, such as product pricing for a particular customer, or whether a customer can return a given product.

In Chapter 10 we will cover EDM in more detail, and describe how the use of Domain Specific Languages (DSLs) can be used to create business-specific, natural language representations of rules most suitable for maintenance by SMEs.

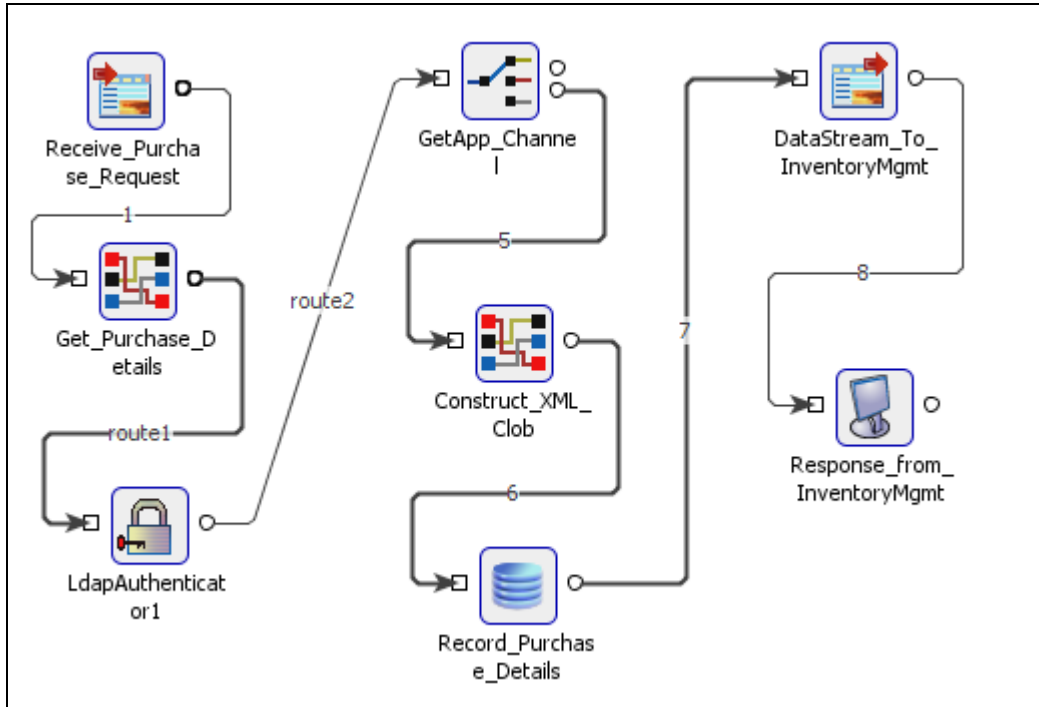
### **1.4.3 Enterprise Service Bus**

An Enterprise Service Bus, or ESB, is at its core a "middleware" application whose role is to provide interoperability between different communication protocols. For example, it is not uncommon for a company to receive incoming ASCII-delimited orders through older protocols such as FTP. An ESB can "lift" that order from the FTP site, transform it into XML, and then submit internally to a web service for consumption/processing.

Please post comments or corrections to the Author online forum at

<http://www.manning-sandbox.com/forum.jspa?forumID=416>

While this can all be done manually, an ESB offers “out-of-the-box” adapters for such processing, and most commonly, event-flow visual modeling tools to generate a chained “microflows”. The cost savings over conventional code techniques is often substantial. Figure 1.10 illustrates how such a data or microflow appears using Fiorano’s ESB:



**Figure 1.10 Example of a Fiorano ESB Data Flow**

How does such data flow differ then a BPM-type application (see figure 1.7)? After all, at first glance, they may appear similar. One key distinction is that BPM applications are typically designed for support of long-running transactions, and use a central orchestration engine to manage how the process flow occurs. A real-time data flow, however, typically uses a model more akin to what is known in choreography. In a choreographed flow, each node (or hop) self-contains the logic of what step to perform next. In addition, a real-time data flow typically passes data by way of message queues, and thus there is a single running instance of the process, with queues corresponding to each node that consume those messages. A BPM, on the other hand, typically instantiates a separate process instance for each new inbound message. This is because, as a potentially long-running transaction, the sequential queuing method would not be appropriate. In order to keep the number of running processes to a reasonable number, a BPM engine will “hydrate” or “dehydrate” the process to-and-from running memory to a serialized form which can then be stored in a database.

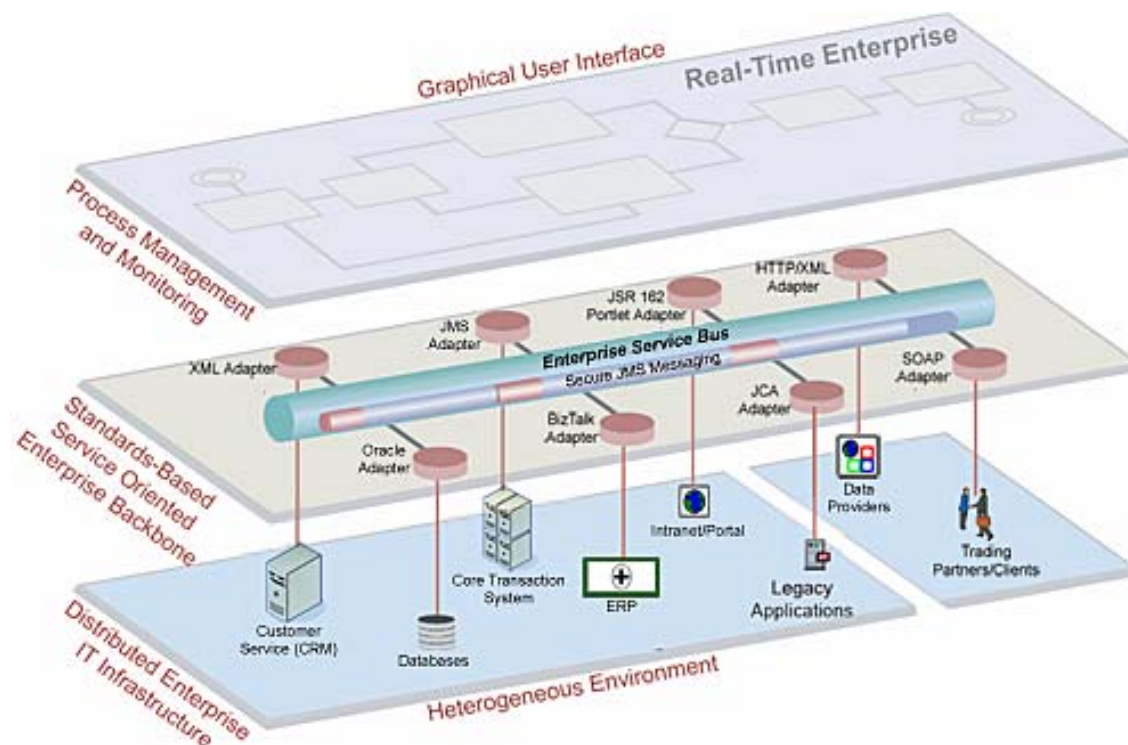
Table 1.1 describes a typical set of services that are provided in an ESB. Because the number of services provided by an ESB, it sometimes is described as a “backplane” or central nervous system that ties together the various SOA technologies

Table 1.1 Core ESB features and capabilities	
FEATURE	DESCRIPTION
Data Connectivity/Adapters	HTTP (SOAP, XML), FTP, SFTP, File and JMS connectivity.

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

- Data Transformation
- Intelligent Routing
- Service Management
- Monitoring & Logging
- Data-flow Choreography
- Custom API
- Timing Services
- XSLT for XML-based transformations.
- Content based-routing based on message properties or in-line XML via XPath. Some include additional, more advanced rule-based routing using a rules engine.
- Administrative tools for managing deployments, versioning and system configuration.
- Ability to monitor, in real-time, document and message flows. Beneficial is capability to put in-line interceptors between nodes, and specifically target individual nodes for more verbose logging.
- Ability to visually (or through editing declarative XML files) create graphs or chains to describe a sequence of steps necessary to complete a data flow.
- The ability to add custom adapters or components to the ESB.
- Ability to create time-based actions or triggers.

.Figure 1.11 depicts the role that an ESB plays in integrating various protocols and how they can be exposed through a standard messaging bus.



**Figure 1.11 Example of an ESB-centric approach for enterprise architecture**

The flexible ability of an ESB to tap into a variety of communication protocols does lend some merit to an ESB-centric architecture. However, if an organization can successfully expose its business services as web services, then the central role that an ESB plays is diminished (under any case, it certainly has a role in an SOA technology stack).

We will now turn our attention to how analytical information can be drawn by the messages that flow through an ESB.

### 1.4.4 Event Stream Processor

Please post comments or corrections to the Author online forum at

<http://www.manning-sandbox.com/forum.jspa?forumID=416>

What is an event? An event is simply something of interest that happens within your business. It may be expected and normal, or abnormal. An event that does not occur may have as much importance as those that do. Too many events may also be indicative of a problem. Why is it relevant to SOA? ESP support can be integrated into the implementation of your services so that real-time visibility into systems becomes a reality. This operational intelligence arms your enterprise with the ability to quickly spot anomalies and respond accordingly. Adding such capabilities into legacy solutions is often not feasible, and instead you must rely upon data warehouse and business intelligence tools, neither of which provide real-time visibility.

Event Stream Processing (ESP) is considered part of a relatively new technology sometimes referred to as Complex Event Processing (CEP). Vivek Ranadive defines it as:

"CEP is an innovative technology that pulls together real-time information from multiple databases, applications and message-based systems and then analyzes this information to discern patterns and trends that might otherwise go unnoticed. CEP gives companies the ability to identify and anticipate exceptions and opportunities buried in seemingly unrelated events". [Ranadive]

The role of an ESP is to receive multiple streams of real-time data and to, in turn, detect patterns among the events. A variety of filters, time-based aggregations, triggers and joins can be typically used by the ESP to assist in pattern detection.

In *Performance Dashboards* [Eckerson] Wayne Eckerson identifies 3 types of business intelligence dashboards: operational, tactical and strategic. *Operational dashboards* are those that generate alerts that notify users about exception conditions. They may also utilize statistical models for predictive forecasting. *Tactical dashboards* provide high-level summary information along with modeling tools. *Strategic dashboards*, as the name implies, are primarily used by executives to ensure company objectives are being met. Operational dashboards rely upon the data that event stream processors generate. As the saying goes, you can't drive while looking in your rearview mirror. For a business to thrive in today's competitive landscape, real-time analysis is essential. This provides a company with the ability to immediately spot cost savings opportunities (such as sudden drops in critical raw materials); proactively identify problem areas (say, a slowdown in web orders due to capacity issues); and unleash new product offerings.

An event architecture strategy must be part of any SOA solution, and must be designed from the get-go to be affective. Bolting on such capabilities later can result in expensive reengineering of code and services. As service components and backbone technologies (such as the ESB) should be propagating notable events. While they a process may not be immediately in place to digest them, adding such capabilities later can be easily introduced by adding new Event Query Language (EQL) expressions into the ESP engine. We will examine EQL in more detail in Chapter 8.

The messages that carry event data that flow into an ESP are, within a Java environment, most likely arrive by way of the Java Messaging Service (JMS), which is addressed next.

### **1.4.5 Java Messaging System**

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

The Java Messaging Service (JMS) is one of the fundamental technologies associated with the Java Platform Enterprise Edition. It is considered “message oriented middleware”(MOM that uses two types of message models: 1) point-to-point queuing model and 2) publish and subscribe model. The queuing model is probably used most frequently, and enables a broadcaster to publish a message to a specific queue, whereby it can then be consumed by a given client. It is considered point-to-point because once the message is consumed by a client, it is not longer available to other clients. A publish/subscribe model is where an event is published to one-or-more interested listeners, or observers. It is model analogous to broadcast TV or Radio, where a publisher (station) is sending out its signal to one or more consumers (listeners).

JMS typically is ideally suited for asynchronous communications, where a “fire-and-forget” paradigm can be used. This contrasts with SOAP-based web services, which follow a request/response type model (this is not a concrete distinction, as there variations of JMS and SOAP which support more than one model, but a generalization). JMS is typically used as one of the enabling technologies within an ESB, and is usually included within that product.

Since JMS is rather ubiquitous in the Java world and well documented through books and articles, it will not be covered directly within this book. It is, however, a critical technology for Java-based SOA environments.

We will now address an often overlooked, but critical technology for building an SOA platform – a registry.

### 1.4.6 Registry

The implementation artifacts that derive from an SOA should be registered within a repository to maximize reuse and provide for management of enterprise assets. Metadata refers to data about data, so in this context, it refers to the properties and attributes of these assets. Assets, as shown in figure 1.12, include service components & composites, business process/orchestrations and applications. It may also include typical LDAP objects such as users, customers and products.

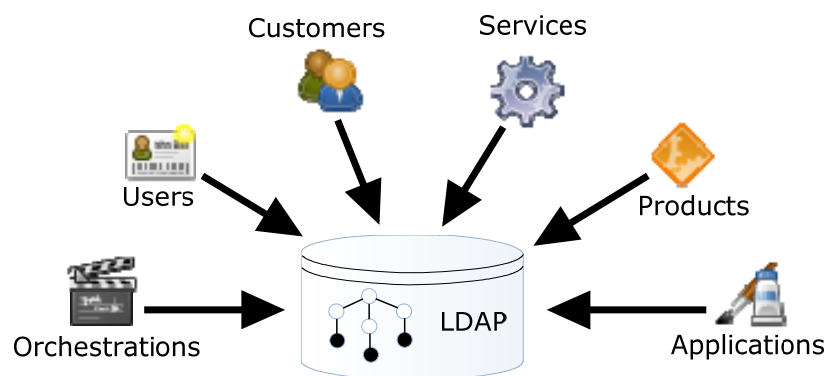


Figure 1.12 Assets that can be managed by an LDAP registry

For smaller organizations, more informal repositories maybe utilized, and could be as simple as Wiki articles or a simple database that describe the various assets. As organizations grown in size, however, having an appropriate technology like LDAP simplifies management and assists in reporting, governance and security profiling. It is

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

important to treat the SOA artifacts as true corporate assets – this represent highly valuable intellectual property, after all.

The metadata attributes for a given asset type will vary, so a flexible repository schema is essential. For example, a service components attributes will include:

- Service endpoint (WS-Addressing)
- Service description
- WSDL location
- Revision/Version #
- Source code location
- Example request/response messages
- Reference to functional and design documents
- Change requests
- Readme files
- Production release records

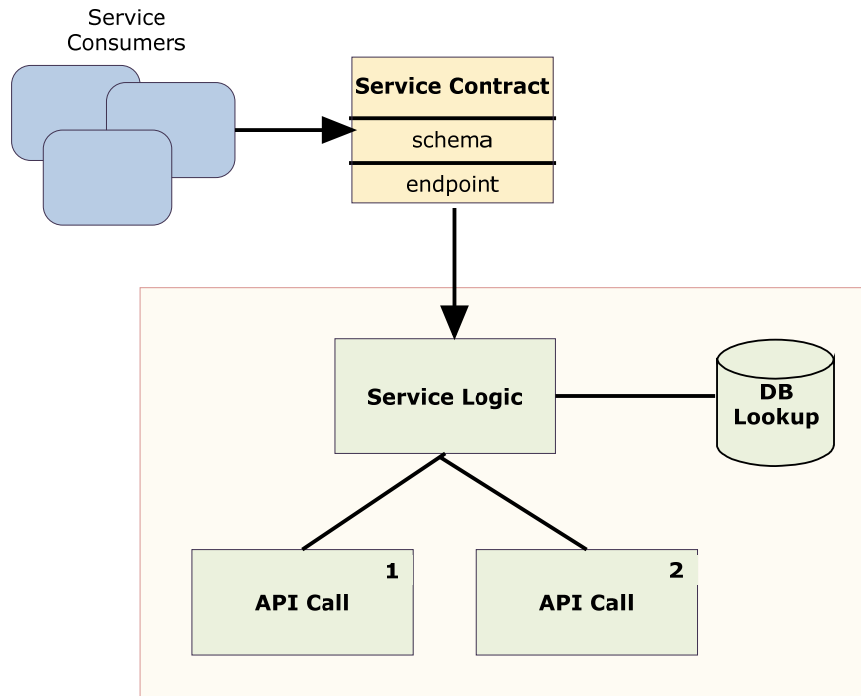
Orchestrations and application may share a similar, if expanded set of attributes, whereas those relating to a user will obviously vary significantly. Chapter 8 will delve much deeper into repositories, with implementation details.

We are nearly completed with our whirlwind overview of critical SOA technologies. One essential, indeed cornerstone to SOA, is addressed next – services.

### **1.4.7 Service Components & Composites**

Service components and composites really represent the core building blocks for what constitutes an SOA platform. A service can be construed as an intelligent business function that combines data and logic to form an abstract interaction with an underlying business service. This service is often a discrete piece of functionality that represents a capability found within an existing application. An example of such a service might be an customer address lookup using information found within a CRM system. The service component, in this instance, “wraps” CRM API calls so that it can be called from a variety of clients using just a customer name as the service input. If the CRM API had to be called directly, a multi-step process of a) first identifying the customerId based on the customer name; b) performing code-list lookups for finding coded values; and c) using the customerId to then call a getAddress operation maybe necessary. The service component abstracts the methods and objects of the CRM into generic method or object and makes the underlying transparent to the calling client. An illustration of such a service facade or wrapper is shown in figure 1.13.

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=416>



**Figure 1.13 Illustration of using a facade/wrapper pattern for exposing service functionality**

There are two fundamental requirements that a service must support: a well defined interface and binding. The interface is the contract which defines the service specification, and is represented as a WSDL for SOAP-based web services. The binding is the communications protocol for how the client will interact with the service. An example of such a protocol would be SOAP over HTTP; JMS, Java Remote Method Invocation (RMI), EJB etc. Using a combination of those two, a developer that wants to create a client that uses a service should be able to do so. Of course, how well the interface is designed will dictate how truly useful the service is.

A composite service, as the name suggests, is created by combining the functionality of one or more individual components. A composite may serve to further abstract functionality, and are often considered “course-grained” services (such as a service to create a new customer). A composite service, in turn, may then be combined with other services to create even higher level composites. In any event, composites share the same requirements as components – and interface and binding.

Thomas Erl classifies compositions into two distinct types: primitive and complex [Erl2007]. A primitive type might be used for simple purposes such as content filtering or routing and usually involves 2-3 individual components. A complex composition could be a BPEL-based service that contains multiple nodes or sequence steps. Chapter 3 will provide in-depth coverage of service components and composites.

Regardless of what protocol and standards your services use, there will likely be scenarios, particularly when integrating with outside organizations, which deviate from your best laid plans. One way to bridge such differences, and as a welcome by-product improve service availability and performance, is through web service mediation technology – the topic of the next section.

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

## 1.4.8 Web Service Mediation

Mediation refers to bridging the differences between two parties. Consistent with that definition, Web Service Mediation (WSM) refers to bridging between different communications protocols, with the result being a SOAP-based web service that can be redirected to an appropriate endpoint. For example, a web mediation engine might be used to process authentication incoming credentials from an external partner's SOAP message using WS-Security (WSS). If approved, the message can then be forwarded, minus the WS-Security heading, to an internal web service to process the request. Or, perhaps a partner is unwilling or unable to use SOAP, and instead prefers a REST (XML over HTTP) solution? Using a mediator, the inbound REST call can be easily transformed into SOAP by adding the appropriate envelope. Even transformations between entirely different protocols, such as FTP to SOAP are typically possible. Figure 1.14 depicts the role of the mediator:

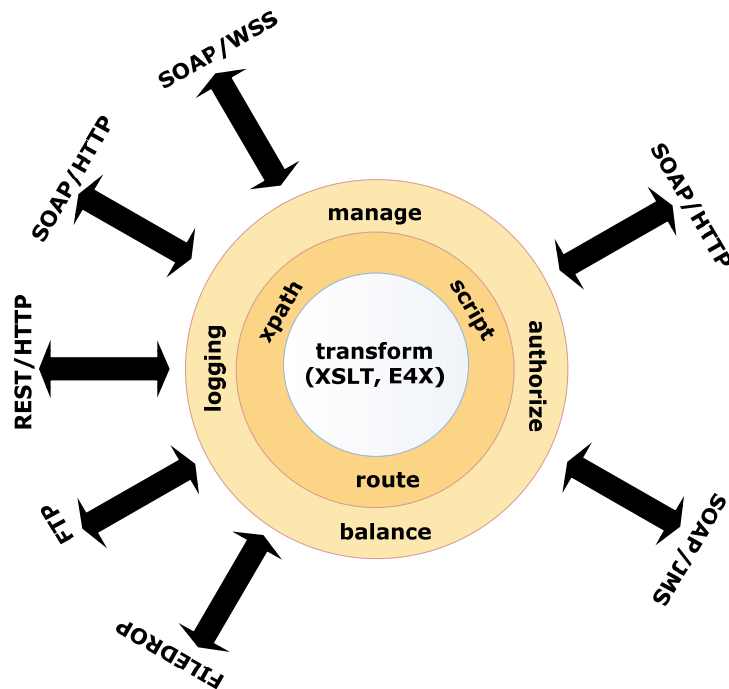


Figure 1.14 Role of web services mediator in bridging between protocols.

A mediator serves other purposes as well, such as: logging all request and responses; internal load balancing; advanced caching; and supporting advanced WS-\* features such as WS-ReliableMessaging. Another important feature is the ability to act as a proxy server. This allows the WSM to transparently intercept outbound messages, log them, and apply a WS-Security envelop, for example. The publisher of the originating message can let such policies be applied externally in a consistent fashion, and not have to worry about implementing such complex details. Compliance and security can be managed independent of the application – a major benefit.

While it is true that ESB can fulfill some of the WSM requirements, they generally do not support the load-balancing, caching and proxy features that WSM provides. As the old English idiom states, use the right “horses for courses”. Or, in more of a laymen’s IT terms, use the right technology in the right places.

Please post comments or corrections to the Author online forum at <http://www.manning-sandbox.com/forum.jspa?forumID=416>

Chapter 5 will address Web Services Mediation in more detail.

## **1.5 Summary**

In this chapter, we've covered the historical origins of SOA, dating back from its roots in prior distributed computing architectures. The emergence of SOAP-based web services is a critical enabler for a SOA, but it turns out that it is only one, albeit critical, part. Simply "exposing" an application's operations as a web service provides little more than earlier RPC-based models. Instead, a deeper dive into what constitutes SOA revealed 5 main technologies and principles that are the bedrock of a SOA environment: service interfaces; service transparency; service loose-coupling and statelessness; service composition; and service registry and publication. With that broad understanding of what constitutes a SOA, we then focused on the technical requirements to form a SOA technology platform. Eight specific technologies were identified that were essential platform building blocks: application server; business process management; enterprise decision management; enterprise service bus; event stream processor; java messaging system; metadata repository; service composition & composites; and web service mediation

Until recently, there hasn't been a robust and complete set of open source technologies that addressed each of these eight areas. Instead, only the commercial vendors, with their deeper pockets and pricy products, appeared able to provide a comprehensive SOA environment. That has changed. Compelling open source solutions now exist for each of those eight technologies, and the next chapter will provide an overview of them. Following that, we will revisit these eight core technologies individually, with substantive examples provided so that you can implement your comprehensive open source SOA platform. The benefits of SOA are no longer limited to big companies with big budgets. Instead, even the smallest of enterprises can participate in this exciting new paradigm by enjoying the fruits of dedicated, and very bright, open source developers. In Chapter 2 we will assess the open source landscape for the SOA technology platform, and identify those which will be the focus for the remainder of the book.

## **1.6 Resources**

### **[BEA]**

BEA White Paper. 2005. "Domain Model for SOA: Realizing the Business Benefit of Service-Oriented Architecture". <http://eudownload.bea.com/uk/events/soa/soa.pdf>

### **[Eckerson]**

Eckerson, Wayne W. 2006. *Performance Dashboards: Measuring, Monitoring and Managing your Business*. Hobokon, NJ: John Wiley & Sons, Inc.

### **[Erl2005]**

Erl, Thomas. 2005. *Service-Oriented Architecture: Concepts, Technology and Design*. Upper Saddle River, NJ: Pearson Education. Pg. 54.

Please post comments or corrections to the Author online forum at

<http://www.manning-sandbox.com/forum.jspa?forumID=416>

**[Erl2007]**

Erl, Thomas. 2007. *SOA: Principles of Service Design*. Boston, MA: Prentice Hall.

**[MargolisSharpe]**

Margolis, Ben and Sharpe, Joseph. 2007. *SOA for the Business Developer. Concepts, BPEL and SCA*. Lewisville, TX: MC Press.

**[Ranadive]**

Ranadive, Vivek. 2006. *Power to Predict*. New York, NY: McGraw-Hill. Pg. 182, 183.

**[SmithFinger]**

Smith, Howard and Finger, Peter. 2003. *Business Process Management – The Third Wave*. Tampa, FL: Meghan-Kiffer Press. Pg. 21

**[TaylorRaden]**

Taylor, James and Raden, Neil. 2007. “*Smart Enough Systems: How to Deliver Competitive Advantage by Automating Hidden Decisions?*”. Boston, MA: Pearson Education, Inc. Pg. 11

Please post comments or corrections to the Author online forum at  
<http://www.manning-sandbox.com/forum.jspa?forumID=416>